

RRRRRRRR	MM	MM	333333	SSSSSSSS	SSSSSSSS	IIIIII	DDDDDDDD	RRRRRRRR
RRRRRRRR	MM	MM	333333	SSSSSSSS	SSSSSSSS	IIIIII	DDDDDDDD	RRRRRRRR
RR RR	RR	MMMM	MMMM	33	33	SS	DD	RR RR
RR RR	RR	MMMM	MMMM	33	33	SS	DD	RR RR
RR RR	RR	MM	MM	33	33	SS	DD	RR RR
RR RR	RR	MM	MM	33	33	SS	DD	RR RR
RRRRRRRR	MM	MM	33	SSSSSS	SSSSSS	IIIIII	DD	RRRRRRRR
RRRRRRRR	MM	MM	33	SSSSSS	SSSSSS	IIIIII	DD	RRRRRRRR
RR RR	RR	MM	MM	33	SS	SS	DD	RR RR
RR RR	RR	MM	MM	33	SS	SS	DD	RR RR
RR RR	RR	MM	MM	33	SS	SS	DD	RR RR
RR RR	RR	MM	MM	33	SS	SS	DD	RR RR
RR RR	RR	MM	MM	333333	SSSSSSSS	SSSSSSSS	IIIIII	DDDDDDDD
RR RR	RR	MM	MM	333333	SSSSSSSS	SSSSSSSS	IIIIII	DDDDDDDD

LL	IIIIII	SSSSSSSS
LL	IIIIII	SSSSSSSS
LL	II	SS
LLLLLLLL	IIIIII	SSSSSSSS
LLLLLLLL	IIIIII	SSSSSSSS

1 0001 0 MODULE RM3SSIDR (LANGUAGE (BLISS32) .
2 0002 0 IDENT = 'V04-000'
3 0003 0) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1 *♦♦
29 0029 1 *♦♦
30 0030 1 *♦♦
31 0031 1 FACILITY: RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1 Search thru a SIDR array and return the first
35 0035 1 non-deleted record
36 0036 1
37 0037 1
38 0038 1 ENVIRONMENT:
39 0039 1
40 0040 1 VAX/VMS OPERATING SYSTEM
41 0041 1
42 0042 1 --
43 0043 1
44 0044 1
45 0045 1 AUTHOR: Todd M. Katz RE-CREATION DATE: 23-Jun-1982
46 0046 1
47 0047 1
48 0048 1 MODIFIED BY:
49 0049 1
50 0050 1 V03-022 RAS0164 Ron Schaefer 29-Jun-1983
51 0051 1 Make references to RMSRU_RECLAIM be LONG_RELATIVE addressing.
52 0052 1
53 0053 1 V03-021 MCN0002 Maria del C. Nasr 24-Mar-1983
54 0054 1 More linkages reorganization
55 0055 1
56 0056 1 V03-020 TMK0013 Todd M. Katz 11-Mar-1983
57 0057 1 Make a change to RMSFOLLOW_PTR so that the primary data bucket

58 0058 1 containing the target record is exclusively accessed if it is
59 0059 1 possible that some reclamation maybe done (the file is write
60 0060 1 accessed and Recovery Unit Journallable).

61 0061 1
62 0062 1 V03-019 MCN0001 Maria del C. Nasr 28-Feb-1983
63 0063 1 Reorganize Linkages

64 0064 1
65 0065 1 V03-018 TMK0012 Todd M. Katz 18-Jan-1983
66 0066 1 Add support for Recovery Unit Journalling and RU ROLLBACK
67 0067 1 Recovery of ISAM files. Support involves modifications to
68 0068 1 the routine RMSFOLLOW_PTR.

69 0069 1
70 0070 1 The purpose of the routines within this module is to find the
71 0071 1 next non-deleted primary data record by means of an alternate
72 0072 1 index. Towards this goal, RMS will search all SIDRs with key
73 0073 1 values matching the key in keybuffer 2 according to the
74 0074 1 characteristics of the search, until such a record is found. If
75 0075 1 during its search RMS encounters records that are marked
76 0076 1 RU_DELETE, RMS will try and delete them for good at this time
77 0077 1 provided it has write access to the file and the Recovery Unit
78 0078 1 in which they were deleted has completed successfully.

79 0079 1
80 0080 1 If RMS is able to delete a primary data record marked RU_DELETE,
81 0081 1 then RMS proceeds to continue looking for a non-deleted primary
82 0082 1 data record just as if it had encountered a deleted record in
83 0083 1 the first place. Likewise, if RMS is unable to delete a record
84 0084 1 that is marked RU_DELETE because it does not have write access
85 0085 1 to the file, it will also continue its search. However, if RMS
86 0086 1 is unable to delete a RU_DELETE marked record for good because
87 0087 1 the Recovery Unit in which it was marked RU_DELETE has not
88 0088 1 successfully terminated, then RMS returns this record as if it
89 0089 1 was the next non-deleted primary data record, and lets a higher
90 0090 1 higher level routine decide whether or not to wait for the
91 0091 1 Recovery Unit in which the record was deleted to complete, or to
92 0092 1 return an error to the user.

93 0093 1
94 0094 1 RMS will also re-format any records that are marked RU_UPDATE
95 0095 1 and are in a special format provided the stream has write access
96 0096 1 to the file, and the Recovery Unit in which the record was
97 0097 1 updated has terminated.

98 0098 1
99 0099 1 In addition to this change, I have made a further enhancement to
100 0100 1 the routine RM\$SEARCH_SIDR. At the present time, when a SIDR
101 0101 1 array element is encountered that should have been marked
102 0102 1 deleted, it is marked deleted without recovering any space if
103 0103 1 and only if the stream has write access to the file, and the
104 0104 1 file is not being shared. I have changed this by removing the
105 0105 1 restriction that to mark a SIDR element deleted the file can not
106 0106 1 be shared. Furthermore, RMS will now not only mark such elements
107 0107 1 as deleted, but reclaim as much space as possible as long as the
108 0108 1 stream has write access to the file.

109 0109 1
110 0110 1 V03-017 TMK0011 Todd M. Katz 05-Jan-1983
111 0111 1 Eliminate the routine RMSFND_SDR_ARRY. This global routine is
112 0112 1 now being called in only one place, and has been folded directly
113 0113 1 into the code.

114 0114 1

: 115 0115 1 | V03-016 TMK0010 Todd M. Katz 04-Jan-1983
: 116 0116 1 | If during positioning to the next primary data record by means
: 117 0117 1 | of an alternate index, RMS encounters a SIDR element that is
: 118 0118 1 | not marked deleted although the primary data record it points
: 119 0119 1 | to is either deleted or the alternate key does not match the
: 120 0120 1 | key of the SIDR, then RMS may mark the element deleted provided
: 121 0121 1 | the file is not being shared and has been open for write access
: 122 0122 1 | before it continues its search for the next primary data record.
: 123 0123 1 | If either of these two conditions is violated, RMS can not mark
: 124 0124 1 | the element deleted, but just positions past it to continue the
: 125 0125 1 | search. What it should also be doing when it positions past the
: 126 0126 1 | element in this case is increment the count of the number of
: 127 0127 1 | preceeding elements in the current SIDR array. RMS was not
: 128 0128 1 | doing this and this represents a bug.
: 129 0129 1 |
: 130 0130 1 | V03-015 TMK0009 Todd M. Katz 07-Dec-1982
: 131 0131 1 | Remove all the re-positioning code. Sigh. This code was
: 132 0132 1 | originally required to fix the 1.5 SIDR deadlock problem. I
: 133 0133 1 | have come up with a different way to fix this problem. I have
: 134 0134 1 | changed how buckets are locked during SDELETEs. A stream can now
: 135 0135 1 | never hold onto a primary data bucket while attempting to
: 136 0136 1 | access a SIDR bucket. Therefore, a stream is able to hold onto
: 137 0137 1 | a SIDR bucket while waiting for a primary data bucket without
: 138 0138 1 | the possibility of a deadlock existing. Thus, there is no
: 139 0139 1 | longer any need to perform any type of SIDR re-positioning, and
: 140 0140 1 | I have eliminated all the new code that used to do it.
: 141 0141 1 |
: 142 0142 1 | V03-014 TMK0008 Todd M. Katz 12-Nov-1982
: 143 0143 1 | The routine RMSFND_SDR_ARRAY requires as input, the size of the
: 144 0144 1 | secondary key it is to search for in IRBSB KEYSZ. This IRAB
: 145 0145 1 | cell might not have the correct key size when this routine is
: 146 0146 1 | called from within RMSSIDR_REPOS. Yet, RMSSIDR_REPOS can not
: 147 0147 1 | merely overwrite the cell with the correct key size because
: 148 0148 1 | RMS might be performing a generic search at the time SIDR
: 149 0149 1 | re-positioning was required, and might later need the value
: 150 0150 1 | stored there. Therefore, RMS must save the value in IRBSB KEYSZ
: 151 0151 1 | when the routine RMSSIDR REPOS is entered, set it to the full
: 152 0152 1 | key size, and restore the original value when the routine is
: 153 0153 1 | exited. if this is not done, the possibility exists that
: 154 0154 1 | RMSFND_SDR_ARRAY might position RMS to the wrong SIDR instead
: 155 0155 1 | of to the SIDR it was positioned to when the routine was first
: 156 0156 1 | entered.
: 157 0157 1 |
: 158 0158 1 | V03-013 TMK0007 Todd M. Katz 09-Nov-1982
: 159 0159 1 | Make another change to the SIDR re-positioning code. IF RMS is
: 160 0160 1 | unable to position either to the primary data record, or back to
: 161 0161 1 | the SIDR during SIDR re-positioning, then before returning RMS
: 162 0162 1 | ascertains whether it was unsuccessful in positioning to the
: 163 0163 1 | current SIDR array, or whether it was unsuccessful at
: 164 0164 1 | positioning even to the SIDR itself. The former case represents
: 165 0165 1 | an error, the latter is reasonable because the SIDR could have
: 166 0166 1 | been deleted and the space it occupied reclaimed. During this
: 167 0167 1 | attempt to re-position to the current SIDR, performed by the
: 168 0168 1 | routine RMSFIND_SDIR, RMS will position either to the current
: 169 0169 1 | SIDR (which would represent an error), or to the next SIDR that
: 170 0170 1 | followed this SIDR. However, if RMS is unable to position to the
: 171 0171 1 | current SIDR and there is no next SIDR (the end-of-file is

172 0172 1
173 0173 1
174 0174 1
175 0175 1
176 0176 1
177 0177 1
178 0178 1
179 0179 1
180 0180 1
181 0181 1
182 0182 1
183 0183 1
184 0184 1
185 0185 1
186 0186 1
187 0187 1
188 0188 1
189 0189 1
190 0190 1
191 0191 1
192 0192 1
193 0193 1
194 0194 1
195 0195 1
196 0196 1
197 0197 1
198 0198 1
199 0199 1
200 0200 1
201 0201 1
202 0202 1
203 0203 1
204 0204 1
205 0205 1
206 0206 1
207 0207 1
208 0208 1
209 0209 1
210 0210 1
211 0211 1
212 0212 1
213 0213 1
214 0214 1
215 0215 1
216 0216 1
217 0217 1
218 0218 1
219 0219 1
220 0220 1
221 0221 1
222 0222 1
223 0223 1
224 0224 1
225 0225 1
226 0226 1
227 0227 1
228 0228 1

encountered), then RMSFIND_SIDR returns an error of RNF. Unfortunately, the routine RMSSIDR REPOS can not return this status to its caller (RMSSEARCH_SIDR) because that would indicate that RMS was unable to find the primary data record during re-positioning - a very grave error and definitely not the case here. When RMSSIDR REPOS returns an error of RNF signalling this very bad condition, RMS converts it into a BUG error, and returns it to the user. However, the error RMS does want to return from RMSSEARCH_SIDR in the case when RMS finds both the primary data record and the SIDR deleted during re-positioning and is unable to position to the next SIDR because of encountering the end-of-file is in fact RNF! In order to arrange things so that this error will be returned under these circumstances, I have changed RMSSIDR REPOS so that it will return an error of EOF when RMSFIND_SIDR returns RNF and RMSSEARCH_SIDR so that it changes the EOF returned by RMSSIDR_REPOS back to RNF.

V03-012 TMK0006 Todd M. Katz 29-Oct-1982
If RMS is successful at finding the target primary data record within RMSFOLLOW_PTR, then save the current index descriptor and replace it with the index descriptor for the primary key before determining record overhead and size of the primary data record. After this determination is made, the descriptor is restored.

V03-011 TMK0005 Todd M. Katz 22-Sep-1982
If a SIDR re-positioning is required, and key compression is enabled, then reset the variable containing the address of the last record in the bucket encountered with a zero front compressed key to be the first record in the bucket. This resetting is only required when RMS must continue the search for a non-deleted primary data record within the current SIDR, and when the possibility exists that the search maybe continued in a SIDR with a completely different key value necessitating its expansion into keybuffer 2. This resetting is necessary because during the re-positioning RMS had to release the SIDR bucket, and then reclaim it, and while it didn't have it locked, anything might have happened to its contents. The reason why the variable is initialized with the address of the first SIDR in the bucket is because the key of the first SIDR must be zero front compressed, and it is the last known record with such a property.

V03-010 KBT0329 Keith B. Thompson 22-Sep-1982
Change check for sharing to sfsb test

V03-009 TMK0004 Todd M. Katz 06-Sep-1982
The field IRBSB_SRCHFLAGS is now a word. Fix all references to it.

The routine RMSSEARCH_SIDR maybe called to search SIDR arrays for a non-deleted primary data record either when RMS is positioning sequentially, or when RMS is positioning randomly by an alternate key value. When I initially wrote this routine I mistakenly wrote it for the sequential case only. Since in this case RMSSEARCH_SIDR only has to search those SIDRs whose keys exactly match the full size search key, this meant that

229 0229 1
230 0230 1
231 0231 1
232 0232 1
233 0233 1
234 0234 1
235 0235 1
236 0236 1
237 0237 1
238 0238 1
239 0239 1
240 0240 1
241 0241 1
242 0242 1
243 0243 1
244 0244 1
245 0245 1
246 0246 1
247 0247 1
248 0248 1
249 0249 1
250 0250 1
251 0251 1
252 0252 1
253 0253 1
254 0254 1
255 0255 1
256 0256 1
257 0257 1
258 0258 1
259 0259 1
260 0260 1
261 0261 1
262 0262 1
263 0263 1
264 0264 1
265 0265 1
266 0266 1
267 0267 1
268 0268 1
269 0269 1
270 0270 1
271 0271 1
272 0272 1
273 0273 1
274 0274 1
275 0275 1
276 0276 1
277 0277 1
278 0278 1
279 0279 1
280 0280 1
281 0281 1
282 0282 1
283 0283 1
284 0284 1
285 0285 1

positioning randomly by key was broken for all cases involving generic search keys, greater-than searches, or greater-than or equal searches when more than one SIDR had to be searched. The fix for this set of problems was actually quite simple. If the search characteristics are not setup such that RMS is performing an exact match search, the routine RM\$SEARCH_SIDR saves the search key in keybuffer 5, and extracts the key of the current SIDR into keybuffer 2. When this SIDR is exhausted and a non-deleted primary data record has not been found, RMS returns the search key saved in keybuffer 5 to keybuffer 2, and determines whether the key of the next SIDR matches the search key according to the characteristics of the search. If so, this whole cycle repeats itself, otherwise, this routine returns the appropriate error to its caller.

I also made a change to RM\$SIDR_REPOS involving the case when RMS is unable to position to a primary data record because it has been deleted, and when it re-positions back to the SIDR it finds that it too has been deleted. In such a case, because RMS was unable to re-position to the SIDR, it did a greater-than or equal search and had positioned exactly to the SIDR that followed. This is exactly where RMS wants to continue its search for a non-deleted primary data record provided the key of this SIDR matches search key according to the search characteristics. Therefore, all the routine RM\$SIDR REPOS has to do is return a 0. The routine RM\$SEARCH_SIDR, when it sees this 0, knows that the current SIDR has been exhausted (or in this case deleted), and that it should go determine whether the search can continue with the SIDR it now finds itself positioned to.

V03-008 KBT0298 Keith B. Thompson 24-Aug-1982
Reorganize psects

V03-007 TMK0003 Todd M. Katz 10-Aug-1982
Change the linkage of RM\$SIDR REPOS. The address of the beginning of the SIDR is now both in the input parameter list and in the output parameter list. This is necessary instead of just passing its address to this routine, an address of a stack location, because this routine will allow RMS to stall, and when RMS resumes after a stall, the stack addresses are not necessarily the same as they were before the stall.

V03-006 TMK0002 Todd M. Katz 10-Aug-1982
When RMS positions to the first non-deleted primary data record by alternate key, it first skips past all entries within a SIDR that are marked deleted until either the end of the SIDR is encountered, or a non-deleted entry is found. Using this non-deleted entry, RMS attempts to position to the primary data record. If RMS finds that the data record is in fact deleted, it wants to mark the SIDR entry deleted, and return to the loop that looks for a non-deleted SIDR entry. There it starts its search for a non-deleted SIDR array element with the current SIDR element which of course RMS has just marked deleted. However, RMS can only mark such a SIDR entry deleted if it has write access to the file and the file is not being shared. Thus, the way this RMS currently works, if the

: 286 0286 1 |
: 287 0287 1 |
: 288 0288 1 |
: 289 0289 1 |
: 290 0290 1 |
: 291 0291 1 |
: 292 0292 1 |
: 293 0293 1 |
: 294 0294 1 |
: 295 0295 1 |
: 296 0296 1 |
: 297 0297 1 |
: 298 0298 1 |
: 299 0299 1 |
: 300 0300 1 |
: 301 0301 1 |
: 302 0302 1 |
: 303 0303 1 |
: 304 0304 1 |
: 305 0305 1 |
: 306 0306 1 |
: 307 0371 1 |
: 308 0372 1 | Define default PSECTS for code
: 309 0373 1 |
: 310 0374 1 | PSECT
: 311 0375 1 | CODE = RMSRMS3(PSECT_ATTR),
: 312 0376 1 | PLIT = RMSRMS3(PSECT_ATTR);
: 313 0377 1 |
: 314 0378 1 | Linkages
: 315 0379 1 |
: 316 0380 1 | LINKAGE
: 317 0381 1 | L_PRESERVE1,
: 318 0382 1 | L_RABREG_567,
: 319 0383 1 | L_RABREG_67,
: 320 0384 1 | L_RABREG_7,
: 321 0385 1 | L_REC_OVHD,
: 322 0386 1 | L_SIDR_FIRST,
: 323 0387 1 | L_JSBOT,
: 324 0388 1 |
: 325 0389 1 | Linkages for Local Routines
: 326 0390 1 |
: 327 0391 1 | RLSFOLLOW_PTR = JSB ()
: 328 0392 1 | : GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN),
: 329 0393 1 | RLSPOS_BY_COUNT = JSB (REGISTER = 1)
: 330 0394 1 | : GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN);
: 331 0395 1 |
: 332 0396 1 | External Routines
: 333 0397 1 |
: 334 0398 1 | EXTERNAL ROUTINE
: 335 0399 1 | RMSCOMPARE_REC : RLSRABREG_67,
: 336 0400 1 | RMSCSEARCH_TREE : RLSRABREG_67,
: 337 0401 1 | RMSEXTR ARRAY RFA : RLSRABREG_67,
: 338 0402 1 | RMSFIND_BY_RRV : RLSRABREG_67,
: 339 0403 1 | RMSKEY_DESC : RLSRABREG_7,
: 340 0404 1 | RMSGETNXT_ARRAY : RLSRABREG_67,
: 341 0405 1 | RMSRECORD_KEY : RLSPRESERVE1,
: 342 0406 1 | RMSREC_OVHD,

```
: 343   0407 1 RMSRLSBKT : RL$PRESERVE1,
: 344   0408 1 RMSRU RECLAIM : RL$RABREG_67 ADDRESSING_MODE(LONG_RELATIVE),
: 345   0409 1 RMSSIDR-END : RL$RABREG_67,
: 346   0410 1 RMSSIDR-FIRST : RLSSIDR FIRST,
: 347   0411 1 RMSQUISH_SIDR : RL$RABREG_567,
: 348   0412 1 RMSUNPACK_REC : RL$JSB01;
: 349   0413 1
: 350   0414 1 ! Forward Routine
: 351   0415 1 !
: 352   0416 1 FORWARD ROUTINE
: 353   0417 1 RM$POS_BY_COUNT : RL$POS_BY_COUNT;
```

```
355 0418 1 XSBTTL 'RMSFOLLOW_PTR'  
356 0419 1 ROUTINE RMSFOLLOW_PTR (VBN, ID) : RLSFOLLOW_PTR =  
357 0420 1 !++  
358 0421 1 FUNCTIONAL DESCRIPTION:  
359 0422 1 This routine receives as input the RFA of an allegedly non-deleted  
360 0423 1 primary data record containing a secondary key of specific value. It  
361 0424 1 positions to that primary data record, and checks whether it is deleted  
362 0425 1 and if not, whether the secondary key whose specific value is stored  
363 0426 1 in keybuffer 2 has been deleted from the primary data record.  
364 0427 1  
365 0428 1 If RMS finds that the target primary data record is marked RU_DELETE  
366 0429 1 and the Recovery Unit in which the record was deleted is still active,  
367 0430 1 then RMS returns positioned to this record and lets a higher level  
368 0431 1 routine decide what to do. If the Recovery Unit in which the record was  
369 0432 1 deleted has successfully terminated, then RMS will return an error of  
370 0433 1 DEL after deleting this RU_DELETED record (if it has write access to  
371 0434 1 the file), and releasing the primary data bucket.  
372 0435 1  
373 0436 1 If RMS encounters a record that is marked RU_UPDATE and is in a special  
374 0437 1 format then RMS will return positioned to this record after  
375 0438 1 reformatting it. The reformatting is done if RMS has write access to  
376 0439 1 the file, and the Recovery Unit in which it was updated has  
377 0440 1 successfully terminated.  
378 0441 1  
379 0442 1  
380 0443 1  
381 0444 1  
382 0445 1  
383 0446 1  
384 0447 1  
385 0448 1  
386 0449 1  
387 0450 1  
388 0451 1  
389 0452 1  
390 0453 1  
391 0454 1  
392 0455 1  
393 0456 1  
394 0457 1  
395 0458 1  
396 0459 1  
397 0460 1  
398 0461 1  
399 0462 1  
400 0463 1  
401 0464 1  
402 0465 1  
403 0466 1  
404 0467 1  
405 0468 1  
406 0469 1  
407 0470 1  
408 0471 1  
409 0472 1  
410 0473 1  
411 0474 1  
CALLING SEQUENCE:  
BSBW RMSFOLLOW_PTR ()  
INPUT PARAMETERS:  
VBN - RFA VBN of the target primary data record  
ID - RFA ID of the target primary data record  
IMPLICIT INPUTS:  
IDX_DFN - address of current index descriptor  
IDXSB_KEYSZ - size of secondary key  
IDXSW_MINRECSZ - minimum size of record to contain key  
IFAB - address of IFAB  
IFBSB_KBUFSZ - size of each keybuffer  
IFBSB_PLG_VER - prologue version of file  
IFBSV_RU - if set, the file is Recovery Unit Journallable  
IFBSV_WRTACC - if set, file is open for write access  
IRAB - address of IRAB  
IRBSL_CURBDB - address of SIDR bucket's BDB  
(0 if re-positioning)  
IRBSL_KEYBUF - address of keybuffers  
IRBSL_RECBUF - address of internal record buffer  
REC_ADDR - address of SIDR array element  
(invalid if re-positioning)
```

```

412      0475 1 | OUTPUT PARAMETERS:
413          0476 1 |     NONE
414
415          0477 1 | IMPLICIT OUTPUTS:
416
417          0478 1 |     IRAB
418          0481 1 |         IRBSL_CURBDB      - address of primary data bucket's BDB (success)
419          0482 1 |         or contents on input (failure)
420          0483 1 |         IRBSL_NXTBDB      - contents on input (success)
421          0484 1 |         or 0 (failure)
422          0485 1 |         IRBSL_RECBUF      - unpacked primary data record
423          0486 1 |         (success and prologue 3 file)
424          0487 1 |         IRBSV_RU_DELETE   - if set, do not reclaim SIDR array space
425          0488 1 |
426          0489 1 |         REC_ADDR        - address of primary data bucket (success)
427          0490 1 |         or contents on input (failure)
428          0491 1 |
429          0492 1 | ROUTINE VALUE:
430          0493 1 |
431          0494 1 |     SUC      - non-deleted primary data record successfully positioned to.
432          0495 1 |     DEL      - primary data record deleted or specific alternate key deleted
433          0496 1 |         from primary data record.
434          0497 1 |     RNF      - primary data record not found.
435          0498 1 |         various I/O errors
436          0499 1 |
437          0500 1 | SIDE EFFECTS:
438          0501 1 |
439          0502 1 |     On success, REC_ADDR points to the primary data record (which will be
440          0503 1 |         unpacked if the file's prologue version is 3), the primary data
441          0504 1 |         bucket's BDB address is stored in IRBSL_CURBDB, and IRBSL_NXTBDB
442          0505 1 |         contains whatever was in IRBSL_CURBDB on input.
443          0506 1 |     On any and all failures, REC_ADDR points to whatever it had pointed to
444          0507 1 |         on input, IRBSL_CURBDB contains whatever it contained on input,
445          0508 1 |         and any buckets accessed have been released.
446          0509 1 |     If the record is marked RU_DELETED, it might have been deleted.
447          0510 1 |     If the record is marked RU_UPDATE or RU_DELETED and RMS is unable to
448          0511 1 |         recover any space from it because the Recovery Unit in which the
449          0512 1 |         record was modified has not completed, then RMS sets the state
450          0513 1 |         bit IRBSV_RU_DELETE whenever it makes a decision to return an error
451          0514 1 |         status of RMSS_DEL and the file has been opened for write access.
452          0515 1 |         This will guarantee that the corresponding SIDR array element will
453          0516 1 |         only be marked RU_DELETE, and no space will be reclaimed from it.
454          0517 1 |     If the record is marked RU_UPDATED, it might have been reformatted.
455          0518 1 |
456          0519 1 | --
457          0520 1 |
458          0521 2 | BEGIN
459          0522 2 |
460          0523 2 | BUILTIN
461          0524 2 |     AP;
462          0525 2 |
463          0526 2 | EXTERNAL REGISTER
464          0527 2 |     COMMON_RAB_STR,
465          0528 2 |     R_IDX_DFN_STR,
466          0529 2 |     R_REC_ADDR_STR;
467          0530 2 |
468          0531 2 | LOCAL

```

```
469      0532 2      SAVE SDR_ADDR,  
470      0533 2      STATUS;  
471      0534 2  
472      0535 2      ! Save the current SIDR array's address, and the address of the SIDR  
473      0536 2      bucket's BDB while performing the primary data record lookup.  
474      0537 2  
475      0538 2      SAVE SDR_ADDR = .REC_ADDR;  
476      0539 2      IRAB[IRBSL_NXTBDB] = .IRAB[IRBSL_CURBDB];  
477      0540 2  
478      0541 2      ! If the file is write accessed and Recovery Unit Journallable, then make  
479      0542 2      sure the primary data bucket containing the target record is exclusively  
480      0543 2      accessed in case reclamation is required.  
481      0544 2  
482      0545 2      IF .IFAB[IFBSV_WRTACC]  
483          AND  
484          .IFAB[IFBSV_RU]  
485      THEN  
486          IRAB[IRBSB_CACHEFLGS] = CSHSM_LOCK;  
487      0549 2  
488      0550 2  
489      0551 2      Position to the primary data record pointing at by the SIDR array element  
490      0552 2      RMS has positioned to.  
491      0553 2  
492      0554 2      STATUS = RMSFIND_BY_RRV (.VBN, .ID, 0);  
493      0555 2  
494      0556 2      ! If RMS is successful at positioning to the target primary data record  
495      0557 2      only to find that it has been deleted within a Recovery Unit, subject  
496      0558 2      this record to further processing before deciding what to do with this  
497      0559 2      record.  
498      0560 2  
499      0561 2      IF .STATUS  
500          AND  
501          .REC_ADDR[IRC$V_RU_DELETE]  
502      THEN  
503          BEGIN  
504              LOCAL  
505                  TEMP_STATUS;  
506  
507          0566 3  
508          0567 3  
509          0568 3  
510          0569 3  
511          0570 3  
512          0571 3  
513          0572 3  
514          0573 3  
515          0574 3  
516          0575 3  
517          0576 4  
518          0577 3  
519          0578 4  
520          0579 4  
521  
522          0580 4  
523          0581 4  
524          0582 4  
525          0583 4  
526          0584 4  
527          0585 4  
528          0586 4  
529          0587 4  
530          0588 4  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
8010  
8011  
8012  
8013  
8014  
8015  
8016  
8017  
8018  
8019  
8020  
8021  
8022  
8023  
8024  
8025  
8026  
8027  
8028  
8029  
8030  
8031  
8032  
8033  
8034  
8035  
8036  
8037  
8038  
8039  
8040  
8041  
8042  
8043  
8044  
8045  
8046  
8047  
8048  
8049  
8050  
8051  
8052  
8053  
8054  
8055  
8056  
8057  
8058  
8059  
8060  
8061  
8062  
8063  
8064  
8065  
8066  
8067  
8068  
8069  
8070  
8071  
8072  
8073  
8074  
8075  
8076  
8077  
8078  
8079  
8080  
8081  
8082  
8083  
8084  
8085  
8086  
8087  
8088  
8089  
8090  
8091  
8092  
8093  
8094  
8095  
8096  
8097  
8098  
8099  
80100  
80101  
80102  
80103  
80104  
80105  
80106  
80107  
80108  
80109  
80110  
80111  
80112  
80113  
80114  
80115  
80116  
80117  
80118  
80119  
80120  
80121  
80122  
80123  
80124  
80125  
80126  
80127  
80128  
80129  
80130  
80131  
80132  
80133  
80134  
80135  
80136  
80137  
80138  
80139  
80140  
80141  
80142  
80143  
80144  
80145  
80146  
80147  
80148  
80149  
80150  
80151  
80152  
80153  
80154  
80155  
80156  
80157  
80158  
80159  
80160  
80161  
80162  
80163  
80164  
80165  
80166  
80167  
80168  
80169  
80170  
80171  
80172  
80173  
80174  
80175  
80176  
80177  
80178  
80179  
80180  
80181  
80182  
80183  
80184  
80185  
80186  
80187  
80188  
80189  
80190  
80191  
80192  
80193  
80194  
80195  
80196  
80197  
80198  
80199  
80200  
80201  
80202  
80203  
80204  
80205  
80206  
80207  
80208  
80209  
80210  
80211  
80212  
80213  
80214  
80215  
80216  
80217  
80218  
80219  
80220  
80221  
80222  
80223  
80224  
80225  
80226  
80227  
80228  
80229  
80230  
80231  
80232  
80233  
80234  
80235  
80236  
80237  
80238  
80239  
80240  
80241  
80242  
80243  
80244  
80245  
80246  
80247  
80248  
80249  
80250  
80251  
80252  
80253  
80254  
80255  
80256  
80257  
80258  
80259  
80260  
80261  
80262  
80263  
80264  
80265  
80266  
80267  
80268  
80269  
80270  
80271  
80272  
80273  
80274  
80275  
80276  
80277  
80278  
80279  
80280  
80281  
80282  
80283  
80284  
80285  
80286  
80287  
80288  
80289  
80290  
80291  
80292  
80293  
80294  
80295  
80296  
80297  
80298  
80299  
80300  
80301  
80302  
80303  
80304  
80305  
80306  
80307  
80308  
80309  
80310  
80311  
80312  
80313  
80314  
80315  
80316  
80317  
80318  
80319  
80320  
80321  
80322  
80323  
80324  
80325  
80326  
80327  
80328  
80329  
80330  
80331  
80332  
80333  
80334  
80335  
80336  
80337  
80338  
80339  
80340  
80341  
80342  
80343  
80344  
80345  
80346  
80347  
80348  
80349  
80350  
80351  
80352  
80353  
80354  
80355  
80356  
80357  
80358  
80359  
80360  
80361  
80362  
80363  
80364  
80365  
80366  
80367  
80368  
80369  
80370  
80371  
80372  
80373  
80374  
80375  
80376  
80377  
80378  
80379  
80380  
80381  
80382  
80383  
80384  
80385  
80386  
80387  
80388  
80389  
80390  
80391  
80392  
80393  
80394  
80395  
80396  
80397  
80398  
80399  
80400  
80401  
80402  
80403  
80404  
80405  
80406  
80407  
80408  
80409  
80410  
80411  
80412  
80413  
80414  
80415  
80416  
80417  
80418  
80419  
80420  
80421  
80422  
80423  
80424  
80425  
80426  
80427  
80428  
80429  
80430  
80431  
80432  
80433  
80434  
80435  
80436  
80437  
80438  
80439  
80440  
80441  
80442  
80443  
80444  
80445  
80446  
80447  
80448  
80449  
80450  
80451  
80452  
80453  
80454  
80455  
80456  
80457  
80458  
80459  
80460  
80461  
80462  
80463  
80464  
80465  
80466  
80467  
80468  
80469  
80470  
80471  
80472  
80473  
80474  
80475  
80476  
80477  
80478  
80479  
80480  
80481  
80482  
80483  
80484  
80485  
80486  
80487  
80488  
80489  
80490  
80491  
80492  
80493  
80494  
80495  
80496  
80497  
80498  
80499  
80500  
80501  
80502  
80503  
80504  
80505  
80506  
80507  
80508  
80509  
80510  
80511  
80512  
80513  
80514  
80515  
80516  
80517  
80518  
80519  
80520  
80521  
80522  
80523  
80524  
80525  
80526  
80527  
80528  
80529  
80530  
80531  
80532  
80533  
80534  
80535  
80536  
80537  
80538  
80539  
80540  
80541  
80542  
80543  
80544  
80545  
80546  
80547  
80548  
80549  
80550  
80551  
80552  
80553  
80554  
80555  
80556  
80557  
80558  
80559  
80560  
80561  
80562  
80563  
80564  
80565  
80566  
80567  
80568  
80569  
80570  
80571  
80572  
80573  
80574  
80575  
80576  
80577  
80578  
80579  
80580  
80581  
80582  
80583  
80584  
80585  
80586  
80587  
80588  
80589  
80590  
80591  
80592  
80593  
80594  
80595  
80596  
80597  
80598  
80599  
80600  
80601  
80602  
80603  
80604  
80605  
80606  
80607  
80608  
80609  
80610  
80611  
80612  
80613  
80614  
80615  
80616  
80617  
80618  
80619  
80620  
80621  
80622  
80623  
80624  
80625  
80626  
80627  
80628  
80629  
80630  
80631  
80632  
80633  
80634  
80635  
80636  
80637  
80638  
80639  
80640  
80641  
80642  
80643  
80644  
80645  
80646  
80647  
80648  
80649  
80650  
80651  
80652  
80653  
80654  
80655  
80656  
80657  
80658  
80659  
80660  
80661  
80662  
80663  
80664  
80665  
80666  
80667  
80668  
80669  
80670  
80671  
80672  
80673  
80674  
80675  
80676  
80677  
80678  
80679  
80680  
80681  
80682  
80683  
80684  
80685  
80686  
80687  
80688  
80689  
80690  
80691  
80692  
80693  
80694  
80695  
80696  
80697  
80698  
80699  
80700  
80701  
80702  
80703  
80704  
80705  
80706  
80707  
80708  
80709  
80710  
80711  
80712  
80713  
80714  
80715  
80716  
80717  
80718  
80719  
80720  
80721  
80722  
80723  
80724  
80725  
80726  
80727  
80728  
80729  
80730  
80731  
80732  
80733  
80734  
80735  
80736  
80737  
80738  
80739  
80740  
80741  
80742  
80743  
80744  
80745  
80746  
80747  
80748
```

```
526      0589  4
527      0590  5
528      0591  4
529      0592  5
530      0593  5
531      0594  5
532      0595  5
533      0596  5
534      0597  5
535      0598  4
536      0599  4
537      0600  4
538      0601  4
539      0602  4
540      0603  4
541      0604  4
542      0605  4
543      0606  3
544      0607  3
545      0608  3
546      0609  3
547      0610  3
548      0611  3
549      0612  3
550      0613  3
551      0614  4
552      0615  3
553      0616  4
554      0617  4
555      0618  4
556      0619  4
557      0620  4
558      0621  4
559      0622  3
560      0623  2
561      0624  2
562      0625  2
563      0626  2
564      0627  2
565      0628  2
566      0629  2
567      0630  2
568      0631  2
569      0632  2
570      0633  2
571      0634  2
572      0635  2
573      0636  2
574      0637  2
575      0638  2
576      0639  2
577      0640  2
578      0641  2
579      0642  2
580      0643  2
581      0644  2
582      0645  2

      | IF .TEMP_STATUS<0,16> NEQU RMSERR(RLK)
      | THEN
      |   BEGIN
      |     STATUS = RMSERR(DEL);
      |
      |     IF .IFAB[IFB$V_WRTACC]
      |     THEN
      |       IRAB[IRB$V_RU_DELETE] = 1;
      |     END;
      |
      |   END
      |
      | The Recovery Unit in which the current record was deleted has
      | completed successfully and RMS was able to reclaim some space
      | from the record. RMS changes the status to RMSS_DEL so that the
      | search will be continued.
      |
      | ELSE
      |   STATUS = RMSERR(DEL);
      |
      | If RMS has discovered that the Recovery Unit in which the target
      | primary data record was deleted has terminated successfully and that
      | the record is in fact deleted, then release the primary data bucket
      | containing the record.
      |
      | IF .STATUS<0,16> EQLU RMSERR(DEL)
      | THEN
      |   BEGIN
      |
      |     GLOBAL REGISTER
      |       R_BDB_STR;
      |
      |     RELEASE (IRAB[IRB$L_CURBDB]);
      |
      |   END;
      |
      | END;
      |
      | If RMS was successful at positioning to the target primary data record
      | only to find that it had been updated but not deleted within a Recovery
      | Unit, and if the file has been opened for write access, then RMS will
      | try to re-format the record provided the Recovery Unit in which the record
      | was modified has completed.
      |
      | IF .STATUS
      |   AND
      |     .IFAB[IFB$V_WRTACC]
      |   AND
      |     .REC_ADDR[IRC$V_RU_UPDATE]
      |   AND
      |     NOT .REC_ADDR[IRC$V_RU_DELETE]
      | THEN
      |   RMSRU_RECLAIM();
      |
      | If RMS was successful at positioning to the primary data record pointed
      | at by the SIDR array element, and the record is NOT marked deleted, then
      | RMS must make sure that it is the right one - ie that the alternate key in
      | this index matches the key of the SIDR which has been saved in
      | keybuffer 2.
```

```
: 583      0646 2      !
: 584      0647 2      IF .STATUS
: 585      0648 2      THEN
: 586      0649 3      BEGIN
: 587      0650 3
: 588      0651 3
: 589      0652 3      LOCAL
: 590      0653 3      RECORD_OVHD,
: 591      0654 3      RECORD_SIZE,
: 592      0655 3      SAVE_UDR_ADDR : REF BBLOCK;
: 593      0656 3      SAVE_UDR_ADDR = .REC_ADDR;
: 594      0657 3
: 595      0658 3      ! Determine the size of the primary data record's overhead, and the
: 596      0659 3      size of the primary data record itself. This is necessary to
: 597      0660 3      determine whether or not the alternate key has been deleted through
: 598      0661 3      truncation of part of the record.
: 599      0662 3
: 600      0663 4      BEGIN
: 601      0664 4
: 602      0665 4      LOCAL
: 603      0666 4      REC_SIZE,
: 604      0667 4      SAVE_IDX_DFN : REF BBLOCK;
: 605      0668 4
: 606      0669 4      SAVE_IDX_DFN = .IDX_DFN;
: 607      0670 4      RMSKEY_DESC(0);
: 608      0671 4
: 609      0672 4      RECORD_OVHD = RMSREC_OVHD(0; REC_SIZE);
: 610      0673 4
: 611      0674 4      ! If the target primary data record was updated within a Recovery
: 612      0675 4      Unit then the true size of the primary data record maybe retrieved
: 613      0676 4      from the last two bytes in the reserved space of the record.
: 614      0677 4
: 615      0678 4      IF .REC_ADDR[IRC$V_RU_UPDATE]
: 616      0679 4      THEN
: 617      0680 5      RECORD_SIZE = .(.REC_ADDR + .RECORD_OVHD
: 618      0681 5      + .REC_SIZE
: 619      0682 4      - IRC$C_DATSZFLD)<0,16>
: 620      0683 4      ELSE
: 621      0684 4      RECORD_SIZE = .REC_SIZE;
: 622      0685 4
: 623      0686 4      IDX_DFN = .SAVE_IDX_DFN;
: 624      0687 3      END;
: 625      0688 3
: 626      0689 3      ! If this is a prologue 3 file then the record must be unpacked in
: 627      0690 3      order to obtain the full size of the record and to be able to compare
: 628      0691 3      the SIDL key with the alternate key in the data record itself. RMS
: 629      0692 3      also sets REC_ADDR to point to the unpacked record, so that the
: 630      0693 3      alternate key may easily be extracted from the record for key
: 631      0694 3      comparison.
: 632      0695 3
: 633      0696 3      NOTE that because RMS always unpacks prologue 3 primary data records
: 634      0697 3      when positioning by a secondary key for key comparison, it is never
: 635      0698 3      necessary to unpack the record a second time during a $GET/$FIND.
: 636      0699 3
: 637      0700 3      IF .IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
: 638      0701 3      THEN
: 639      0702 4      BEGIN
```

```
: 640      0703 4
: 641      0704 4
: 642      0705 4      BUILTIN
: 643      0706 4      AP:
: 644      0707 4
: 645      0708 4      GLOBAL REGISTER
: 646      0709 4      R_BKT_ADDR;
: 647      0710 4      REC_ADDR = .REC_ADDR + .RECORD_OVHD;
: 648      0711 4
: 649      0712 4      AP = 0;
: 650      0713 4      RECORD_SIZE = RMSUNPACK_REC (.IRAB[IRBSL_RECBUF], .RECORD_SIZE);
: 651      0714 4
: 652      0715 4      REC_ADDR = .IRAB[IRBSL_RECBUF];
: 653      0716 3      END;
: 654      0717 3
: 655      0718 3      | If the primary data record is of sufficient size to contain the
: 656      0719 3      alternate key (ie - the alternate key has not been deleted through
: 657      0720 3      truncation of the record during the process of updating the record
: 658      0721 3      by some other process), then RMS determines whether or not the
: 659      0722 3      alternate key in the record still matches the SIDR it has positioned
: 660      0723 3      to. If this is the case then RMS has found the next non-deleted
: 661      0724 3      primary data record, and returns success.
: 662      0725 3
: 663      0726 4      IF (.RECORD_SIZE GEQU .IDX_DFN[IDX$W_MINRECSZ])
: 664      0727 3      AND
: 665      0728 4      (NOT RM$COMPARE_REC (KEYBUF_ADDR(2), .IDX_DFN[IDX$B_EYSZ], -1))
: 666      0729 3      THEN
: 667      0730 4      BEGIN
: 668      0731 4      REC_ADDR = .SAVE_UDR_ADDR;
: 669      0732 4      RETURN RMSSUC();
: 670      0733 4      END
: 671      0734 4
: 672      0735 4      | If the secondary key has been deleted totally from the record through
: 673      0736 4      truncation of the record during an $UPDATE by another process, or
: 674      0737 4      if during the $UPDATE the secondary key changed, and the secondary
: 675      0738 4      key in the primary data record no longer matches the key of the SIDR
: 676      0739 4      RMS is positioned to, then this is not the next non-deleted primary
: 677      0740 4      data record. In this case, RMS releases the primary data bucket, and
: 678      0741 4      sets up to return an error status of record deleted, so that the
: 679      0742 4      search for the next non-deleted primary data record can continue.
: 680      0743 4
: 681      0744 3      ELSE
: 682      0745 4      BEGIN
: 683      0746 4
: 684      0747 4      GLOBAL REGISTER
: 685      0748 4      R_BDB_STR;
: 686      0749 4
: 687      0750 4      | If the target primary data record has been modified within a
: 688      0751 4      Recovery Unit, and the stream has write access to the file so
: 689      0752 4      that some attempt will be made to reclaim the space occupied by
: 690      0753 4      SIDR array element, then set up so that when the SIDR array
: 691      0754 4      element pointing to this primary data record is deleted, it is
: 692      0755 4      only marked RU_DELETE, and no space is reclaimed.
: 693      0756 4
: 694      0757 4      IF .IFAB[IFBSV_WRTACC]
: 695      0758 4      AND
: 696      0759 5      (.SAVE_UDR_ADDR[IRC$V_RU_DELETE]
```

```

: 697      0760 5          OR
: 698      0761 5          .SAVE_UDR_ADDR[IRC$V_RU_UPDATE])
: 699      0762 4          THEN
: 700      0763 4          IRAB[IRBSV_RU_DELETE] = 1;
: 701      0764 4
: 702      0765 4          RELEASE(IRAB[IRB$L_CURBDB]);
: 703      0766 4          STATUS = RMSERR(DEC);
: 704      0767 3          END;
: 705      0768 3
: 706      0769 2          END;
: 707      0770 2
: 708      0771 2          ! If RMS was unable to position to the target primary data record, or if
: 709      0772 2          after positioning to the record it found that the record itself, or the
: 710      0773 2          secondary key it expected to find there had been deleted, then RMS
: 711      0774 2          restores the address of the SIDR bucket's BDB to IRB$L_CURBDB, and the
: 712      0775 2          address of the SIDR array element it currently has positioned to
: 713      0776 2          to REC_ADDR, and it returns the appropriate status.
: 714      0777 2
: 715      0778 2          IRAB[IRB$L_CURBDB] = .IRAB[IRB$L_NXTBDB];
: 716      0779 2          IRAB[IRB$L_NXTBDB] = 0;
: 717      0780 2          REC_ADDR = .SAVE_SDR_ADDR;
: 718      0781 2
: 719      0782 2          RETURN .STATUS;
: 720      0783 2
: 721      0784 1          END;

```

! { of routine }

```

.TITLE RM3SSIDR
.IDENT \V04-000\

.EXTRN RMSCOMPARE REC, RMSSEARCH_TREE
.EXTRN RMSEXT ARR$ RFA
.EXTRN RMSFIND_BY_RRV, RMSKEY_DESC
.EXTRN RMSGETNXT_ARRAY
.EXTRN RMSRECORD_KEY, RMSREC_OVHD
.EXTRN RMSRLSBKT, RM$RU RECLAIM
.EXTRN RMSSIDR END, RMSSIDR FIRST
.EXTRN RMSSQUISH_SIDR, RMSUNPACK_REC

.PSECT RMSRMS3,NOWRT, GBL, PIC,2

```

				3C BB 00000 RMSFOLLOW_PTR:			
				3C A9 0A 40 A9	20 06 56 DD 00002	PUSHR #^M<R2,R3,R4,R5>	0419
					A9 D0 00004	PUSHL REC ADDR	0538
					AA E9 00009	MOVL 32(IRAB), 60(IRAB)	0539
					01 E1 0000D	BLBC 6(IFAB), 1\$	0545
					01 90 00013	BBC #1, 160(IFAB), 1\$	0547
					7E D4 00017	MOV#1, 64(IRAB)	0549
					1\$: CLRL -(SP)		0554
					20 AE DD 00019	PUSHL ID	
					20 AE DD 0001C	PUSHL VBN	
					0000G 30 0001F	BSBW RMSFIND_BY_RRV	
					OC C0 00022	ADDL2 #12, SP	
					50 D0 00025	MOVL R0, STATUS	
					53 E9 00028	BLBC STATUS, 5\$	
					05 E1 0002B	BBC #5 (REC ADDR), 4\$	
					EF 16 0002F	JSB RM\$RU_RECLAIM	
					0000000G		

RM3SS1DR
V04-000

RMSFOLLOW_PTR

E 10
16-Sep-1984 02:07:52 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 13:01:41 DISKS\$VMSMASTER:[RMS.SRC]RM

Page 15
32:1 (2)

RM
VO

82AA	16		50	E8 00035	BLBS	TEMP_STATUS, 2\$		0590
	8F		50	B1 00038	CMPW	TEMP_STATUS, #33450		
	53	8262	14	13 0003D	BEQL	3\$		0593
	0B	06	AA	E9 00044	MOVZWL	#33378, STATUS		
07	A9		20	88 00048	BLBC	6(IFAB), 3\$		0595
			05	:1 0004C	BISB2	#32, 7(IRAB)		0597
					BRB	3\$		0576
8262	53	8262	8F	3C 0004E	MOVZWL	#33378, STATUS		0607
	8F		53	B1 00053	CMPW	STATUS, #33378		0614
	54	20	0F	12 00058	BNEQ	4\$		
		20	A9	D0 0005A	MOVL	32(IRAB), BDB		0621
			A9	D4 0005E	CLRL	32(IRAB)		
			7E	D4 00061	CLRL	-(SP)		
			0000G	30 00063	BSBW	RMSRLSBKT		
	5E		04	C0 00066	ADDL2	#4, SP		
	12		53	E9 00069	BLBC	STATUS, 5\$		0631
0A	0E	06	AA	E9 0006C	BLBC	6(IFAB), 5\$		0633
06	66		06	E1 00070	BBC	#6, (REC_ADDR), 5\$		0635
	66		05	E0 00074	BBS	#6, (REC_ADDR), 5\$		0637
	03	00000000G	EF	16 00078	JSB	RMSRU RECLAIM		0639
			53	E8 0007E	BLBS	STATUS, 6\$		0647
			0094	31 00081	BRW	13\$		
	52		56	D0 00084	MOVL	REC_ADDR, SAVE_UDR_ADDR		0656
	55		57	D0 00087	MOVL	IDX_DFN, SAVE_IDX_DFN		0669
			7E	D4 0008A	CLRL	-(SP)		0670
			0000G	30 0008C	BSBW	RMSKEY_DESC		
	5E		04	C0 0008F	ADDL2	#4, SP		
			51	D4 00092	CLRL	R1		0672
			0000G	30 00094	BSBW	RMSREC_OVHD		
0D	66		06	E1 00097	BBC	#6, (REC_ADDR), 7\$		0678
54	56		50	C1 0009B	ADDL3	RECORD_OVHD, REC_ADDR, R4		0680
	54	FE A144	9F	0009F	PUSHAB	-2(REC_SIZE)[R4]		
			9E	3C 000A3	MOVZWL	@(SP)+, RECORD_SIZE		
			03	11 000A6	BRB	8\$		
	54		51	D0 000A8	MOVL	REC_SIZE, RECORD_SIZE		0684
	57		55	D0 000AB	MOVL	SAVE_IDX_DFN, IDX_DFN		0686
	03	00B7	CA	91 000AE	CMPB	183(IFAB), #3		0700
			16	1F 000B3	BLSSU	9\$		
	56		50	C0 000B5	ADDL2	RECORD_OVHD, REC_ADDR		0710
			5C	D4 000B8	CLRL	AP		0712
	51		54	D0 000BA	MOVL	RECORD_SIZE, R1		0713
	50	68	A9	D0 000BD	MOVL	104(IRAB), R0		
		0000G	30	000C1	BSBW	RMSUNPACK_REC		
	54		50	D0 000C4	MOVL	R0, RECORD_SIZE		
	56	68	A9	D0 000C7	MOVL	104(IRAB), REC_ADDR		0715
			00	ED 000CB	CMPZV	#0, #16, \$4(IDX_DFN), RECORD_SIZE		0726
			21	1A 000D1	BGTRU	10\$		
	7E		01	CE 000D3	MNEGL	#1, -(SP)		0728
	7E	20	A7	9A 000D6	MOVZBL	32(IDX_DFN), -(SP)		
	50	00B4	CA	3C 000DA	MOVZWL	180(IFAB), R0		
			60	B940	PUSHAB	96(IRAB)[R0]		
			0000G	30 000E3	BSBW	RMSCOMPARE_REC		
	5E		0C	C0 000E6	ADDL2	#12, SP		
	08		50	E8 000E9	BLBS	R0, 10\$		
	56		52	D0 000EC	MOVL	SAVE_UDR_ADDR, REC_ADDR		0731
	50		01	D0 000EF	MOVL	#1, R0		0732
			32	11 000F2	BRB	14\$		

RM3SSIDR
V04-000

RMSFOLLOW_PTR

F 10
16-Sep-1984 02:07:52 VAX-11 Bliss-32 v4.0-742
14-Sep-1984 13:01:41 DISK\$VMSMASTER:[RMS.SRC]RM3SSIDR.B32:1

Page 16
(2)

RM
VC

04	0C	06	AA	E9	000F4	10\$:	BLBC	6(IFAB), 12\$: 0757	
	62	05	E0	000F8			BBS	#5, (SAVE_UDR_ADDR), 11\$: 0759	
04	62	06	E1	000FC			BBC	#6, (SAVE_UDR_ADDR), 12\$: 0761	
	07	A9	20	88	00100	11\$:	BISB2	#32, 7(IRAB)	: 0763	
		54	20	A9	00	00104	12\$:	MOVL	32(IRAB), BDB	: 0765
			20	A9	D4	00108	CLRL	32(IRAB)		
				7E	D4	0010B	CLRL	-(SP)		
				0000G	30	0010D	BSBW	RMSRLSBKT		
		5F	8262	04	C0	00110	ADDL2	#4, SP		
		53		8F	3C	00113	MOV?WL	#33378, STATUS	: 0766	
	20	A9	3C	A9	00	00118	13\$:	MOVL	60(IRAB), 32(IRAB)	: 0778
			3C	A9	D4	0011D	CLRL	60(IRAB)	: 0779	
		56		6E	D0	00120	MOVL	SAVE_SDR_ADDR, REC_ADDR	: 0780	
		50		53	D0	00123	MOVL	STATUS, R0	: 0782	
		5E		04	C0	00126	14\$:	ADDL2	#4, SP	: 0784
			3C	BA	00129		POPR	#^M<R2,R3,R4,R5>		
				05	0012B		RSB		:	

; Routine Size: 300 bytes, Routine Base: RMSRMS3 + 0000

```
723      0785 1 %SBTTL 'RMSPOS_BY_COUNT'
724      0786 1 ROUTINE RMSPOS_BY_COUNT (SKIP_NUMBER) : RL$POS_BY_COUNT =
725      0787 1 ++
726      0788 1
727      0789 1 FUNCTIONAL DESCRIPTION:
728      0790 1
729      0791 1 This routine's responsibility is to position to a particular SIDR
730      0792 1 array element. It does this by skipping SKIP_NUMBER array elements.
731      0793 1
732      0794 1 CALLING SEQUENCE:
733      0795 1
734      0796 1     BSBW RMSPOS_BY_COUNT()
735      0797 1
736      0798 1 INPUT PARAMETERS:
737      0799 1
738      0800 1     SKIP_NUMBER      - number of array elements to skip
739      0801 1
740      0802 1 IMPLICIT INPUTS:
741      0803 1
742      0804 1     REC_ADDR          - address of SIDR
743      0805 1
744      0806 1 OUTPUT PARAMETERS:
745      0807 1     NONE
746      0808 1
747      0809 1 IMPLICIT OUTPUTS:
748      0810 1
749      0811 1     REC_ADDR          - address of desired array element
750      0812 1
751      0813 1 ROUTINE VALUE:
752      0814 1
753      0815 1     SUC      - positioned to desired array element.
754      0816 1     BUG      - the number of array elements preceding the desired
755      0817 1           element exceeds the number of elements in the entire array.
756      0818 1
757      0819 1 SIDE EFFECTS:
758      0820 1
759      0821 1
760      0822 1 -- on success, REC_ADDR points to the desired array element.
761      0823 1
762      0824 2 BEGIN
763      0825 2
764      0826 2 EXTERNAL REGISTER
765      0827 2     COMMON_RAB_STR,
766      0828 2     R_IDX_DFN_STR,
767      0829 2     R_REC_ADDR_STR;
768      0830 2
769      0831 2 LOCAL
770      0832 2     END_OF_SIDR;
771      0833 2
772      0834 2     ! Find the end of the current SIDR.
773      0835 2
774      0836 2     END_OF_SIDR = RMSSIDR_END();
775      0837 2
776      0838 2     ! Position to the SIDR's first array element
777      0839 2
778      0840 2     REC_ADDR = RMSSIDR_FIRST(0);
779      0841 2
```

```

780 0842 2 | Skip the desired number of array elements, or until the end of the SIDR
781 0843 2 | 's encountered - which ever comes first.
782 0844 2
783 0845 2 DECRU LOOP_INDEX FROM .SKIP_NUMBER TO 1 BY 1
784 0846 2 DO
785 0847 2
786 0848 2 | If the end of the SIDR is encountered before the desired array
787 0849 2 | element is encountered then this represents an invalid internal
788 0850 2 | condition, and report it as such.
789 0851 2
790 0852 2 IF .REC_ADDR GEQA .END_OF_SIDR
791 0853 2 THEN
792 0854 2     RETURN RMSERR(BUG)
793 0855 3
794 0856 3 | Position past the current array element to the next array element.
795 0857 3
796 0858 2 ELSE
797 0859 2     RMSGETNXT_ARRAY();
798 0860 2
799 0861 2 | Having positioned to the desired SIDR array element, return success.
800 0862 2
801 0863 2 RETURN RMSSUC();
802 0864 1 END:           ! { of routine }

```

	1C BB 00000 RM\$POS_BY COUNT:		
53	51 D0 00002	POSHR #^M<R2,R3,R4>	0786
54	0000G 30 00005	MOVL R1 R3	0836
	50 D0 00008	BSBW RM\$SIDR-END	
	7E D4 0000B	MOVL R0, END_OF_SIDR	0840
	0000G 30 0000D	CLRL -(SP)	
5E	04 C0 00010	BSBW RM\$SIDR-FIRST	
56	50 D0 00013	ADDL2 #4, SP	
52	53 D0 00016	MOVL R0, REC_ADDR	0852
	11 11 00019	MOVL SKIP_NUMBER, LOOP_INDEX	
54	56 D1 0001B 1\$:	BRB 3\$	
	07 1F 0001E	CMPL REC_ADDR, END_OF_SIDR	
50	8434 8F 3C 00020	BLSSU 2\$	0854
	0A 11 00025	MOVZWL #33844, R0	
	0000G 30 00027 2\$:	BRB 4\$	0859
	52 D7 0002A	BSBW RMSGETNXT_ARRAY	
	ED 12 0002C 3\$:	DECL LOOP_INDEX	0852
50	01 D0 0002E	BNEQ 1\$	
	1C BA 00031 4\$:	MOVL #1, R0	0863
	05 00033	POPR #^M<R2,R3,R4>	
		RSB	0864

: Routine Size: 52 bytes. Routine Base: RMSRMS3 + 012C

804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860

0865 1 %SBTTL 'RMSSEARCH_SIDR'
0866 1 GLOBAL ROUTINE RMSSEARCH_SIDR : RLSRABREG_67 =
0867 1 //++
0868 1
0869 1
0870 1 FUNCTIONAL DESCRIPTION:
0871 1
0872 1 The purpose of this routine is to search for a non-deleted primary
0873 1 data record by means of an alternate key of reference. This routine
0874 1 starts its search with an array element in the current SIDR array. If
0875 1 RMS is positioning sequentially, then the array element RMS starts
0876 1 its search with will always be one more than the value in
0877 1 IRBSW_SAVE_POS. Thus, RMS may start its search with any of the elements
0878 1 in the current SIDR array depending upon the value in this variable.
0879 1 However, if RMS is positioning randomly by key value, then RMS will
0880 1 always start its search for a non-deleted primary data record with the
0881 1 very first element in the current SIDR.
0882 1
0883 1 When the current SIDR array is exhausted, RMS will attempt to continue
0884 1 its search within the very next SIDR array that follows. RMS may
0885 1 continue its search provided the key of this next SIDR matches the
0886 1 characteristics of the search. These search characteristics include the
0887 1 search key itself (in keybuffer 2), the size of the search key (in
0888 1 IRBSB_KEYSZ) which may not be the same as the size of the alternate key
0889 1 itself, and whether the search is for an equal match, a greater-than
0890 1 or equal match (IRBSV_SRCHGE is set), or a greater-than match
0891 1 (IRBSV_SRCHGT is set). If RMS is positioning randomly then any possible
0892 1 combination of these search characteristics is possible, but when RMS is
0893 1 positioning sequentially, RMS will only search those SIDRs whose key
0894 1 exactly matches the full size search key in keybuffer 2.
0895 1
0896 1 If during its search for a non-deleted primary data record RMS
0897 1 encounters a non-deleted SIDR array element pointing to a deleted
0898 1 primary data record, then RMS will mark the SIDR array element deleted
0899 1 and reclaim as much space from the SIDR as it can according to the
0900 1 normal rules of SIDR deletion. However, such activity will only take
0901 1 place if the stream has write access to the file.
0902 1
0903 1 This routine concludes its search when either an error occurs, it is
0904 1 successful at positioning to a non-deleted primary data record, or it
0905 1 exhausts all SIDRs matching the search characteristics without
0906 1 positioning to such a primary data record.
0907 1
0908 1
0909 1
0910 1
0911 1
0912 1
0913 1
0914 1
0915 1
0916 1
0917 1
0918 1
0919 1
0920 1
0921 1

CALLING SEQUENCE:
BSBW RMSSEARCH_SIDR()

INPUT PARAMETERS:
NONE

IMPLICIT INPUTS:
IDX_DFN - address of index descriptor describing key
IDXSV_KEY COMPRESS - if set, SIDR key compression is enabled
IDXSB_KEYSZ - size of key
IFAB - address of IFAB

861 0922 1 | IFB\$L_SFSB PTR - pointer to first shared file sync. block
 862 0923 1 | IFB\$W_KBUFSZ - size of a key buffer
 863 0924 1 | IFBSV_WRTACC - if set, file is write accessed
 864 0925 1 |
 865 0926 1 | IRAB - address of IRAB
 866 0927 1 | IRBSL_CURBDB - address of SIDR bucket's BDB
 867 0928 1 | IRBSL_KEYBUF - pointer to the contiguous keybuffers
 868 0929 1 | IRBSL_KEYSZ - size of search key
 869 0930 1 | IRBSL_LST_NCMP - address of last zero front compressed key
 870 0931 1 | IRBSV_PRM - if set, permanence should be set in SIDR BDB
 871 0932 1 | IRBSW_SAVE_POS - number of elements preceding starting point
 872 0933 1 | IRBSV_SRCHGE - if set, search for GE match
 873 0934 1 | IRBSV_SRCHGT - if set, search for GT match
 874 0935 1 |
 875 0936 1 | REC_ADDR - address of SIDR within bucket
 876 0937 1 |
 877 0938 1 | OUTPUT PARAMETERS:
 878 0939 1 | NONE
 879 0940 1 |
 880 0941 1 | IMPLICIT OUTPUTS:
 881 0942 1 |
 882 0943 1 | IRAB - address of IRAB
 883 0944 1 | IRBSL_CURBDB - address of primary data bucket's BDB
 884 0945 1 | IRBSW_FIRST_ID - RFA ID of new current SIDR's first element
 885 0946 1 | IRBSL_FIRST_VBN - RFA VBN of new current SIDR's first element
 886 0947 1 | IRBSL_LST_NCMP - address of last zero front compressed key
 887 0948 1 | IRBSL_RFA_VBN - VBN of new current SIDR's bucket
 888 0949 1 | IRBSW_SAVE_POS - number of elements before new current element
 889 0950 1 |
 890 0951 1 | REC_ADDR - address of primary data record
 891 0952 1 |
 892 0953 1 | ROUTINE VALUE:
 893 0954 1 |
 894 0955 1 | BUG - invalid internal condition
 895 0956 1 | SUC - next non-deleted primary data record found
 896 0957 1 | RNF - all SIDRs of this key value exhausted
 897 0958 1 | various I/O errors
 898 0959 1 |
 899 0960 1 | SIDE EFFECTS:
 900 0961 1 |
 901 0962 1 | AP is trashed.
 902 0963 1 | Keybuffer 5 will contain the search key, if this routine was called
 903 0964 1 | with a generic search key (search key size less than key size),
 904 0965 1 | or to find the first SIDR with a key greater-than or
 905 0966 1 | greater-than-or-equal to the search key in keybuffer 2.
 906 0967 1 | On success, IRBSL_CURBDB contains the primary data bucket's BDB
 907 0968 1 | address, the SIDR bucket will have been released, REC_ADDR
 908 0969 1 | points to the primary data bucket, and the IRAB contains
 909 0970 1 | sufficient information necessary to uniquely identify and
 910 0971 1 | re-position to the SIDR array element pointing to the
 911 0972 1 | non-deleted primary data record. In addition, keybuffer 2 will
 912 0973 1 | contain the key of the SIDR.
 913 0974 1 | On all errors except for a RNF found error, the SIDR bucket will have
 914 0975 1 | been released and possibly marked dirty/permanent.
 915 0976 1 | SIDR array elements and in fact whole SIDRs might have been deleted
 916 0977 1 | if the stream positioning has write access to the file.
 917 0978 1 |

```
RM$SEARCH_SIDR          0979 1 !--  
918 0980 1 BEGIN  
919 0981 2 BUILTIN  
920 0982 2 AP  
921 0983 2 TESTBITS:  
922 0984 2 EXTERNAL REGISTER  
923 0985 2 COMMON RAB_STR,  
924 0986 2 R_IDX_BFN_STR,  
925 0987 2 R_REC_ADDR_STR;  
926 0988 2  
927 0989 2  
928 0990 2  
929 0991 2  
930 0992 2  
931 0993 2  
932 0994 2  
933 0995 2  
934 0996 2  
935 0997 2  
936 0998 2  
937 0999 2 : BLOCK [1],  
938 1000 2 STATUS;  
939 1001 2  
940 1002 2  
941 1003 2 FIRST_TIME = 0,0,1,0 %;  
942 1004 2  
943 1005 2 NEW_KEY = 0,1,1,0 %;  
944 1006 2  
945 1007 2  
946 1008 2 Save the address of the SIDR, and the address of the first byte past the  
947 1009 2 last SIDR array element (ie the end of the SIDR).  
948 1010 2  
949 1011 2 BEG_OF_SIDR = .REC ADDR;  
950 1012 2 END_OF_SIDR = RMSSIDR_END();  
951 1013 2  
952 1014 2 RMS must perform an initial positioning before it begins its search for  
953 1015 2 the next non-deleted primary data record. This positioning depends upon  
954 1016 2 whether RMS is positioning randomly by key value, or is positioning  
955 1017 2 sequentially. In the latter case, where RMS initially positions to will  
956 1018 2 depend upon whether RMS had been able to position to the current SIDR  
957 1019 2 before calling this routine. Thus, this routine is faced with four  
958 1020 2 different ways in which it could perform this initial positioning  
959 1021 2 depending upon the mode of access, and if the access mode is sequential,  
960 1022 2 whether or not RMS had positioned to the current SIDR before this  
961 1023 2 routine was called.  
962 1024 2  
963 1025 2 1. If RMS is positioning randomly by key value then RMS has positioned to  
964 1026 2 the first SIDR whose key matches the search key according to the  
965 1027 2 search characteristics, and RMS wants to begin its search for a  
966 1028 2 non-deleted primary data record with the very first element in the  
967 1029 2 SIDR array.  
968 1030 2  
969 1031 2 2. This is the first positioning attempt after a $REWIND or $CONNECT and  
970 1032 2 as it is sequential there is no current SIDR. In this case RMS has  
971 1033 2 positioned to the very first SIDR it could find, and begins its search  
972 1034 2 for the next non-deleted primary data record with the very first  
973 1035 2 element in the SIDR array.
```

975 1036 2 | 3. RMS has successfully positioned to the current SIDR. In this case RMS
976 1037 2 begins its search for the next non-deleted primary data record with
977 1038 2 the first element that follows the current SIDR element. Note that
978 1039 2 there may not be a following element in which case RMS exhausts the
979 1040 2 current SIDR very quickly.
980 1041 2
981 1042 2 | 4. RMS was unable to position to the current SIDR before calling this
982 1043 2 routine. This could only be because the entire current SIDR was
983 1044 2 deleted. In this case RMS has positioned to the first SIDR with a
984 1045 2 key value greater than or equal to the key of the current SIDR, and
985 1046 2 it begins its search for the next non-deleted primary data record with
986 1047 2 the first array element in this SIDR.
987 1048 2
988 1049 2 RM\$POS_BY_COUNT (.IRAB[IRBSW_SAVE_POS]);
989 1050 2
990 1051 2 | If RMS is positioning randomly by key value and either the size of the
991 1052 2 search key is less than the true key size or the search is for the first
992 1053 2 record with an alternate key either greater-than-or-equal or greater-than
993 1054 2 that of the search key, then save the search key presently located in
994 1055 2 keybuffer 2 in keybuffer 5. It is necessary to save the search key because
995 1056 2 if the SIDR positioned to is exhausted without locating a non-deleted
996 1057 2 primary data record, it will be necessary to continue the search with the
997 1058 2 next SIDR matching the search characteristics, and the search key will be
998 1059 2 required in order to continue this search. In addition, it will be
999 1060 2 necessary each time RMS positions to a new SIDR to extract its key into
1000 1061 2 keybuffer 2, since the key of the SIDR RMS has currently positioned to
1001 1062 2 might not be the same as the keys of SIDRs RMS has previously positioned
1002 1063 2 to during the course of its search for a non-deleted primary data record.
1003 1064 2
1004 1065 2 IF .IRAB[IRBSV_SRCHGE]
1005 1066 2 OR
1006 1067 2 .IRAB[IRBSV_SRCHGT]
1007 1068 2 OR
1008 1069 2 .IRAB[IRBSB_KEYSZ] LSSU .IDX_DFN[IDX\$B_KEYSZ]
1009 1070 2 THEN
1010 1071 3 BEGIN
1011 1072 3 FLAGS[NEW_KEY] = 1;
1012 1073 3 CH\$MOVE (:IRAB[IRBSB_KEYSZ], KEYBUF_ADDR(2), KEYBUF_ADDR(5));
1013 1074 3 END
1014 1075 3
1015 1076 3 | Regardless of how many SIDRs RMS will position to in its quest for a
1016 1077 3 non-deleted primary data record, their key value will always be the
1017 1078 3 same as the key in keybuffer 2 (once keybuffer 2 has a key if it doesn't
1018 1079 3 already), and there will not be a need to extract the key out of each
1019 1080 3 SIDR positioned to.
1020 1081 3
1021 1082 2 ELSE
1022 1083 2 FLAGS[NEW_KEY] = 0;
1023 1084 2
1024 1085 2 | Continue until either the next non-deleted primary data record is
1025 1086 2 encountered, the end-of-file is reached, or some error is encountered.
1026 1087 2
1027 1088 2 NEXT:
1028 1089 3 BEGIN
1029 1090 3 WHILE 1
1030 1091 3 DO
1031 1092 3

```
1032      1093 4      BEGIN
1033      1094 4
1034      1095 4      ! If RMS is currently positioned to the very first element of a
1035      1096 4      S IDR array, and either this is first such array being scanned
1036      1097 4      during this current positioning or the search characteristics
1037      1098 4      indicate that the key of each S IDR positioned to must be
1038      1099 4      extracted, then extract the key of the S IDR into keybuffer 2.
1039      1100 4
1040      1101 5      IF (TESTBITS(FLAG$FIRST_TIME)
1041          OR
1042          .FLAG$NEW_KEY)
1043          AND
1044          (.IRAB[IRBSW_SAVE_POS] EQLU 0)
1045
1046      1107 4      THEN
1047      1108 5      BEGIN
1048      1109 5
1049      1110 5      LOCAL
1050      1111 5          TMP_REC_ADDR;
1051      1112 5
1052      1113 5          TMP_REC_ADDR = .REC_ADDR;
1053      1114 5          REC_ADDR = .BEG_OF_S IDR;
1054      1115 5
1055      1116 5          REC_ADDR = .REC_ADDR + RMSREC_OVHD(-1);
1056      1117 5
1057      1118 5          IF .IDX_DFN[IDX$V_KEY_COMP]
1058      1119 5          THEN
1059      1120 6              BEGIN
1060      1121 6
1061      1122 6              GLOBAL REGISTER
1062      1123 6                  R_BDB;
1063      1124 6
1064      1125 6
1065      1126 6          AP = 1;
1066      1127 6          RMSRECORD_KEY (KEYBUF_ADDR(2));
1067      1128 5
1068      1129 5          END
1069      1130 5          ELSE
1070      1131 5              CHSMOVE (.IDX_DFN[IDX$B_KEYSZ], .REC_ADDR, KEYBUF_ADDR(2));
1071      1132 4
1072      1133 4
1073      1134 4      ! Continue searching for the next non-deleted primary data record
1074      1135 4      until either it is found, an error occurs, the end of the S IDR
1075      1136 4      is encountered, or no S IDR with a key matching the search key
1076      1137 4      according to the search characteristics remains to be scanned for an
1077      1138 4      array element pointing to a non-deleted primary data record.
1078      1139 4
1079      1140 4      WHILE .REC_ADDR LSSA .END_OF_S IDR
1080      1141 4      DO
1081      1142 5          BEGIN
1082      1143 5
1083      1144 5          ! Starting with the current S IDR array element, search for the
1084      1145 5          first element in the S IDR array that is not marked deleted
1085      1146 5          ! until either one is found or the end of the S IDR is encountered.
1086      1147 5
1087      1148 7          WHILE ((.REC_ADDR LSSA .END_OF_S IDR)
1088      1149 6              AND
```

```
1089      1150 6          .REC_ADDR[IRC$V_DELETED])  
1090      1151 5          DO BEGIN  
1091      1152 6  
1092      1153 6  
1093      1154 6          RMSGETNXT_ARRAY();  
1094      1155 6          IRAB[IRBSW_SAVE_POS] = .IRAB[IRBSW_SAVE_POS] + 1;  
1095      1156 5          END;  
1096      1157 5  
1097      1158 5          ! If RMS has not runoff the end of the current SIDR array it is  
1098      1159 5          searching, but has successfully positioned to an element in the  
1099      1160 5          SIDR array that is not marked deleted, then attempt to access the  
1100      1161 5          primary data record that it points to, and determine whether or  
1101      1162 5          not it is marked deleted.  
1102      1163 5  
1103      1164 5          IF .REC_ADDR LSSA .END_OF_SIDR  
1104      1165 5          THEN BEGIN  
1105      1166 6  
1106      1167 6  
1107      1168 6  
1108      1169 6          LOCAL  
1109      1170 6          RFA_ID,  
1110      1171 6          RFA_VBN;  
1111      1172 6          ! Attempt to access the primary data record while holding on to  
1112      1173 6          the SIDR bucket lock; waiting for the lock on the primary data  
1113      1174 6          bucket to be released if someone else has it while holding  
1114      1175 6          onto the SIDR bucket lock if necessary.  
1115      1176 6  
1116      1177 6          RMSEXT_ARRY_RFA (RFA_VBN, RFA_ID);  
1117      1178 6  
1118      1179 6          STATUS = RMSFOLLOW_PTR (.RFA_VBN, .RFA_ID);  
1119      1180 6  
1120      1181 6          ! If RMS has successfully found the next non-deleted primary  
1121      1182 6          data record, or some serious error has been encountered,  
1122      1183 6          then terminate the search for the next non-deleted primary  
1123      1184 6          data record. If the error encountered was just that the  
1124      1185 6          primary data record accessed had been deleted, then RMS will  
1125      1186 6          be able to continue the search for the next non-deleted  
1126      1187 6          primary data record with the next element in the SIDR array  
1127      1188 6          currently being searched which is not marked deleted.  
1128      1189 6  
1129      1190 7          IF .STATUS<0,16> NEQU RMSERR(DEL)  
1130      1191 6          THEN LEAVE NEXT;  
1131      1192 6  
1132      1193 6  
1133      1194 6          ! If the primary data record positioned to is deleted, but the  
1134      1195 6          SIDR array element is not marked as such, then alleviate this  
1135      1196 6          discrepancy provided the user has accessed the file for  
1136      1197 6          writing, before continuing on with the search for the next  
1137      1198 6          non-deleted primary data record. If the current SIDR array  
1138      1199 6          element is marked RU_DELETE, and RMS is to RU_DELETE it, then  
1139      1200 6          in this case there is no need to do anything.  
1140      1201 6  
1141      1202 6          IF .IFAB[IFBSV_WRTACC]  
1142      1203 6          AND  
1143      1204 7          NOT (.REC_ADDR[IRC$V_RU_DELETE]  
1144      1205 7          AND  
1145      1206 7          .IRAB[IRBSV_RU_DELETE])
```

```
: 1146      1207  6          THEN
: 1147      1208  7          BEGIN
: 1148      1209  7          GLOBAL REGISTER
: 1149      1210  7          COMMON_IO_STR;
: 1150      1211  7          LOCAL
: 1151      1212  7          SAVE_REC_ADDR;
: 1152      1213  7          ! Retrieve the address of the SIDR bucket and mark it dirty.
: 1153      1214  7          BDB = .IRAB[IRBSL_CURBDB];
: 1154      1215  7          BDB[BDB$V_DRT] = T;
: 1155      1216  7          BKT_ADDR = .BDB[BDB$L_ADDR];
: 1156      1217  7          ! Save the address of the current SIDR array element before
: 1157      1218  7          attempting to delete any space.
: 1158      1219  7          SAVE_REC_ADDR = .REC_ADDR;
: 1159      1220  7          REC_ADDR[IRC$V_RU_DELETE] = 0;
: 1160      1221  7          ! If RMS is able to only delete the space occupied by the
: 1161      1222  7          current SIDR array element's RRV pointer, then RMS may
: 1162      1223  7          continue its search for a non-deleted primary data record
: 1163      1224  7          within the current SIDR array after re-determining the
: 1164      1225  7          address of the end of the current SIDR array.
: 1165      1226  7          IF RMSSQUISH_SIDR (1, .BEG_OF_SIDR)
: 1166      1227  7          THEN
: 1167      1228  7          IF .REC_ADDR EQLU .SAVE_REC_ADDR
: 1168      1229  7          THEN
: 1169      1230  7          BEGIN
: 1170      1231  7          REC_ADDR = .BEG_OF_SIDR;
: 1171      1232  7          END_OF_SIDR = RM$SIDR-END();
: 1172      1233  7          REC_ADDR = .SAVE_REC_ADDR;
: 1173      1234  7          END
: 1174      1235  7          ! If RMS is able to delete the entire current SIDR
: 1175      1236  7          because all the elements contained within it have
: 1176      1237  7          been deleted, then leave the loop responsible for
: 1177      1238  8          searching the current SIDR so that conditions might
: 1178      1239  8          be set up to attempt to search the next SIDR array
: 1179      1240  8          ;
: 1180      1241  8          ;
: 1181      1242  8          ;
: 1182      1243  8          ;
: 1183      1244  8          ;
: 1184      1245  8          ;
: 1185      1246  8          ;
: 1186      1247  8          ;
: 1187      1248  8          ;
: 1188      1249  8          ;
: 1189      1250  8          ;
: 1190      1251  7          ;
: 1191      1252  7          ELSE
: 1192      1253  7          EXITLOOP
: 1193      1254  7          ! If RMS is able to only mark the current SIDR array as
: 1194      1255  7          deleted without recovering any space then make sure the
: 1195      1256  7          state bit IRBSV_RU_DELETE is clear (if it was set) and
: 1196      1257  7          continue the search for a non-deleted primary data record
: 1197      1258  7          with the next SIDR array element.
: 1198      1259  7          ;
: 1199      1260  7          ELSE
: 1200      1261  8          BEGIN
: 1201      1262  8          IRAB[IRBSV_RU_DELETE] = 0;
: 1202      1263  8          RMSGETNXT_ARRAY();
```

```
: 1203      1264  8
: 1204      1265  7
: 1205      1266  7
: 1206      1267  7
: 1207      1268  7
: 1208      1269  7
: 1209      1270  7
: 1210      1271  7
: 1211      1272  7
: 1212      1273  7
: 1213      1274  7
: 1214      1275  7
: 1215      1276  7
: 1216      1277  7
: 1217      1278  7
: 1218      1279  7
: 1219      1280  6
: 1220      1281  7
: 1221      1282  7
: 1222      1283  7
: 1223      1284  7
: 1224      1285  6
: 1225      1286  5
: 1226      1287  5
: 1227      1288  4
: 1228      1289  4
: 1229      1290  4
: 1230      1291  4
: 1231      1292  4
: 1232      1293  4
: 1233      1294  4
: 1234      1295  4
: 1235      1296  4
: 1236      1297  4
: 1237      1298  4
: 1238      1299  4
: 1239      1300  4
: 1240      1301  4
: 1241      1302  4
: 1242      1303  4
: 1243      1304  4
: 1244      1305  4
: 1245      1306  4
: 1246      1307  4
: 1247      1308  4
: 1248      1309  4
: 1249      1310  4
: 1250      1311  4
: 1251      1312  4
: 1252      1313  3
: 1253      1314  3
: 1254      1315  3
: 1255      1316  2
: 1256      1317  2
: 1257      1318  2
: 1258      1319  2
: 1259      1320  2

      IRAB[IRBSW_SAVE_POS] = .IRAB[IRBSW_SAVE_POS] + 1;
      END;
      ! If the file has not been opened for write access or there are
      ! really no changes that need to be made then the SIDR element
      ! can not be marked deleted, and RMS must immediately position
      ! to the next SIDR array element. This is because having
      ! unexpectedly encountered a deleted primary data record RMS
      ! will want to continue its search for the next non-deleted
      ! SIDR element. Since RMS always starts such a search with the
      ! current element, and since RMS was unable to mark the current
      ! SIDR array element deleted, RMS would end up positioning to
      ! the current element as if it was the next element.
      ! Immediately positioning to the next element prevents this.

      ELSE
        BEGIN
          RM$GETNXT_ARRAY();
          IRAB[IRBSW_SAVE_POS] = .IRAB[IRBSW_SAVE_POS] + 1;
          IRAB[IRBSV_RU_DELETE] = 0;
        END;
      END;

      ! Having exhausted this current SIDR array in the search for the next
      ! non-deleted primary data record, RMS positions to the next SIDR
      ! in order to continue the search. NOTE that if the search
      ! characteristics require the original search key, it is restored to
      ! keybuffer 2 from keybuffer 5 before initiating the search. This
      ! will always be the case when the key of the next SIDR RMS positions
      ! to maybe different from the key of the current SIDR.

      IF .FLAG$NEW_KEY
      THEN
        CH$MOVE (.IRAB[IRBSB_KEYSZ], KEYBUF_ADDR(5), KEYBUF_ADDR(2));
        IRAB[IRBSV_NORLS_RNF] = 1;
        RETURN_ON_ERROR TRMSCSEARCH_TREE(), IRAB[IRBSV_PRM] = 0;
        ! Reset all the fields necessary to search a SIDR array starting
        ! from its first element.
        BEG_OF_SIDR = .REC_ADDR;
        END_OF_SIDR = RM$SIDR_END();
        REC_ADDR = RM$SIDR_FIRST(0);
        IRAB[IRBSW_SAVE_POS] = 0;
      END;

      END;                                ! {of NEXT}

      ! If some serious error was encountered in RMS's search for the next
      ! non-deleted primary data record such that the search could no longer
      ! be continued, then RMS releases any bucket locks the process has
```

: 1260 1321 2 | outstanding (primary or SIDR but not both), and returns the error.
: 1261 1322 2 |
: 1262 1323 2 | There is one error case which is handled differently. A record not found
: 1263 1324 2 | error on a search for a primary data record is a serious error. To signal
: 1264 1325 2 | to a caller of this routine that in this particular case this routine
: 1265 1326 2 | should never be recalled, RMS returns an invalid internal condition or
: 1266 1327 2 | bug error after storing the RNF status in the RAB's RAB\$L_STV field.
: 1267 1328 2 |
: 1268 1329 2 IF NOT .STATUS
: 1269 1330 2 THEN BEGIN
: 1270 1331 3 |
: 1271 1332 3 | GLOBAL REGISTER
: 1272 1333 3 | COMMON_IO_STR;
: 1273 1334 3 |
: 1274 1335 3 |
: 1275 1336 3 | IF .IRAB[IRB\$L_CURBDB] NEQU 0
: 1276 1337 3 | THEN RELEASE (.IRAB[IRB\$L_CURBDB]);
: 1277 1338 3 |
: 1278 1339 3 |
: 1279 1340 4 | IF .STATUS<0,16> EQLU RMSERR(RNF)
: 1280 1341 3 | THEN BEGIN
: 1281 1342 4 | | RAB[RAB\$L_STV] = .STATUS;
: 1282 1343 4 | | STATUS = RMSERR(BUG);
: 1283 1344 4 | | END;
: 1284 1345 3 |
: 1285 1346 3 |
: 1286 1347 3 | IRAB[IRB\$V_PRM] = 0;
: 1287 1348 3 | RETURN .STATUS;
: 1288 1349 3 | END
: 1289 1350 3 |
: 1290 1351 3 | If RMS has been successful in positioning to the next non-deleted
: 1291 1352 3 | primary data record, then save all information necessary to position
: 1292 1353 3 | to this SIDR array element as if it was the current element (because it
: 1293 1354 3 | soon will be), release the SIDR bucket, and return success.
: 1294 1355 3 |
: 1295 1356 2 | ELSE BEGIN
: 1296 1357 3 | |
: 1297 1358 3 | | GLOBAL REGISTER
: 1298 1359 3 | | COMMON_IO_STR;
: 1299 1360 3 | |
: 1300 1361 3 | | LOCAL
: 1301 1362 3 | | SAVE_REC_ADDR;
: 1302 1363 3 | |
: 1303 1364 3 | |
: 1304 1365 3 | | BDB = .IRAB[IRB\$L_NXTBDB];
: 1305 1366 3 | | IRAB[IRB\$L_NXTBDB] = 0;
: 1306 1367 3 | |
: 1307 1368 3 | | IRAB[IRB\$L_RFA_VBN] = .BDB[BDB\$L_VBN];
: 1308 1369 3 | |
: 1309 1370 3 | | SAVE_REC_ADDR = .REC_ADDR;
: 1310 1371 3 | | REC_ADDR = .BEG_OF_SIDR;
: 1311 1372 3 | | RMS\$IDR FIRST (T; IRAB[IRB\$L_FIRST_VBN], IRAB[IRB\$W_FIRST_ID]);
: 1312 1373 3 | | REC_ADDR = .SAVE_REC_ADDR;
: 1313 1374 3 |
: 1314 1375 3 |
: 1315 1376 3 | IF .IRAB[IRB\$V_PRM]
: 1316 1377 3 | THEN

1317	1378	3	BDBLBDBSV_PRM] = 1;
1318	1379	3	
1319	1380	3	RM\$RLSBKT(0);
1320	1381	3	IRAB[IRBSV_PRM] = 0;
1321	1382	3	
1322	1383	3	RETURN RMSSUC();
1323	1384	2	END;
1324	1385	2	
1325	1386	1	END;

. {of the routine RMSSEARCH_SIDR}

3C BB 00000 RM\$SEARCH SIDR::									
									0866
08	5E		1C	C2 00002	POSHR	#^M<R2,R3,R4,R5>			
08	AE		01	88 00005	SUBL2	#28, SP			1005
04	AE		56	00 00009	BISB2	#1, FLAGS			1010
		0000G	30	0000D	MOVL	REC_ADDR, BEG_OF_SIDR			1011
			50	00 00010	BSBW	RM\$SIDR_END			
	6E		A9	3C 00013	MOVL	R0, END_OF_SIDR			
	51	76	B3	10 00017	MOVZWL	118(IRAB), R1			1049
0D	42	A9	04	E0 00019	BSBB	RM\$POS_BY_COUNT			
08	42	A9	01	E0 0001E	BBS	#4, 66(IRAB), 1\$			1065
20	A7	00A6	C9	91 00023	BBS	#1, 66(IRAB), 1\$			1067
			1A	1E 00029	CMPB	166(IRAB), 32(IDX_DFN)			1069
	08	AE	02	88 0002B	BGEQU	2\$			
	51	00A6	C9	9A 0002F	BISB2	#2, FLAGS			1072
	50	00B4	CA	3C 00034	MOVZBL	166(IRAB), R1			1073
9E	60	B940	60	B940 DF 00039	MOVZWL	180(IFAB), R0			
			51	28 0003D	PUSHAL	@96(IRAB)[R0]			
			04	11 00043	MOVCL3	R1, @96(IRAB)[R0], a(SP)+			
					BRB	3\$			
05	08	AE	02	8A 00045	2\$: BICB2	#2, FLAGS			1065
44	08	AE	00	E4 00049	3\$: BBSC	#0, FLAGS, 4\$			1083
	08	AE	01	E1 0004E	BBC	#1, FLAGS, 7\$			1101
		76	A9	B5 0005	TSTW	118(IRAB)			1103
			3F	12 00056	BNEQ	7\$			1105
	0C	AE	56	D0 00058	MOVL	REC_ADDR, TMP_REC_ADDR			
	56	04	AE	D0 0005C	MOVL	BEG_OF_SIDR, REC_ADDR			1113
	51		01	CE 00060	MNEG	#1, R1			1114
		0000G	30	00063	BSBW	RM\$REC_OVHD			1116
14	56		50	C0 00066	ADDL2	R0, REC_ADDR			
	1C	A7	06	E1 00069	BBC	#6, 28(IDX_DFN), 5\$			1118
	5C		01	D0 0006E	MOVL	#1, AP			1125
	50	00B4	CA	3C 00071	MOVZWL	180(IFAB), R0			1126
		60	B940	9F 00076	PUSHAB	@96(IRAB)[R0]			
		0000G	30	0007A	BSBW	RM\$RECORD_KEY			
	5E		04	C0 0007D	ADDL2	#4, SP			
			11	11 00080	BRG	6\$			
	51		20	A7 9A 00082	5\$: MOVZBL	32(IDX_DFN), R1			1118
	50	00B4	CA	3C 00086	MOVZWL	180(IFAB), R0			1129
	50	60	A9	C0 0008B	ADDL2	96(IRAB), R0			
	66		51	28 0008F	MOVC3	R1, (REC_ADDR), (R0)			
	55	OC	AE	D0 00093	6\$: MOVL	TMP_REC_ADDR, REC_ADDR			1131
60	6E		56	D1 00097	7\$: CMPL	REC_ADDR, END_OF_SIDR			1140
			03	1F 0009A	BLSSU	8\$			

		6E	0094 31 0009C 56 D1 0009F 8\$: F3 1E 000A2 02 F1 000A4 0000G 30 000A8 76 A9 B6 000AB EF 11 000AE E5 1E 000B0 9\$: 14 AE 9F 000B2 1C AE 9F 000B5 0000G 30 000B8 08 C0 000BB 14 AE DD 000BE 1C AE DD 000C1 FDD9 30 000C4 08 C0 000C7 03 13 000D4 009F 31 000D6 06 AA E9 000D9 10\$: 05 E1 000DD 05 E0 000E1 20 A9 D0 000E6 11\$: 02 88 000EA 18 A4 D0 000EE 56 D0 000F2 20 8A 000F5 04 AE DD 000F8 01 DD 000FB 0000G 30 000FD 08 C0 00100 14 50 E9 00103 52 56 D1 00106 28 12 00109 56 04 AE D0 0010B 0000G 30 0010F 6E 50 D0 00112 56 52 D0 00115 16 11 00118 07 A9 20 8A 0011A 12\$: 0000G 30 0011E 76 A9 B6 00121 0A 11 00124 0000G 30 00126 13\$: 76 A9 B6 00129 20 8A 0012C FF64 31 00130 14\$: 01 E1 00133 15\$: 00A6 C9 9A 00138 51 50 00B4 CA 3C 0013D 60 B940 DF 00142 9E 51 28 00146 42 A9 20 88 0014C 16\$: 0000G 30 00150 07 50 E8 00153 42 A9 80 8F 8A 00156	BRW CMPL BGEQU BBC BSBW INCW BRB BGEQU PUSHAB PUSHAB BSBW ADDL2 PUSHL PUSHL BSBW ADDL2 MOVL CMPW BEQL BRW BLBC BBC BBS MOVL BISB2 MOVL MOVL BICB2 PUSHL PUSHL BSBW ADDL2 BLBC RO, 12\$ CMPL BNEQ MOVL BSBW BSBW MOVL MOVL BRB BEG_OF_SIDR RMSSQUISH_SIDR #8, SP RO, 12\$ REC_ADDR, SAVE_REC_ADDR 15\$ BEG_OF_SIDR, REC_ADDR RMSSIDR_END RO, END_OF_SIDR SAVE_REC_ADDR, REC_ADDR 14\$ #32, 7(IRAB) RMSGETNXT_ARRAY INCW BRB 14\$ RMSGETNXT_ARRAY INCW 118(IRAB) BICB2 BEG_OF_SIDR #32, 7(IRAB) BRW #1, FLAGS, 16\$ MOVZBL MOVZWL PUSHAL MOVCL R1 @(SP)+, @96(IRAB)[R0] BISB2 BSBW BLBS BICB2	15\$ REC_ADDR, END_OF_SIDR 7\$ #2, (REC_ADDR), 9\$ RMSGETNXT_ARRAY 118(IRAB) 8\$ 7\$ RFA_ID RFA_VBN RMSEXT_ARRY_RFA #8, SP RFA_ID RFA_VBN RMSFOLLOW_PTR #8, SP RO, STATUS STATUS, #33378 10\$ 18\$ 6(IFAB), 13\$ #5, (REC_ADDR), 11\$ #5, 7(IRAB), 13\$ 32(IRAB), BDB #2, 10(BDB) 24(BDB), BKT_ADDR REC_ADDR, SAVE_REC_ADDR #32, (REC_ADDR) BEG_OF_SIDR #1 #8, SP RO, 12\$ REC_ADDR, SAVE_REC_ADDR 15\$ BEG_OF_SIDR, REC_ADDR RMSSIDR_END RO, END_OF_SIDR SAVE_REC_ADDR, REC_ADDR 14\$ #32, 7(IRAB) RMSGETNXT_ARRAY INCW 118(IRAB) BEG_OF_SIDR #32, 7(IRAB) #32, 7(IRAB) 7\$ #1, FLAGS, 16\$ 166(IRAB), R1 180(IFAB), RO @96(IRAB)[R0] R1 @(SP)+, @96(IRAB)[R0] #32, 66(IRAB) RMSSEARCH_TREE STATUS, 17\$ #128, 66(IRAB)	1148 1150 1154 1155 1148 1164 1177 1179 1190 1202 1204 1206 1218 1219 1220 1225 1226 1234 1236 1239 1240 1241 1236 1262 1263 1264 1202 1282 1283 1284 1164 1298 1300 1302 1303
--	--	----	---	---	---	--

04	AE	4F	11	0015B		BRB	21\$		1308
		56	D0	0015D	17\$:	MOVL	REC_ADDR, BEG_OF_SIDR		1309
		0000G	30	00161		BSBW	RMS\$IDR_END		
		6E	D0	00164		MOVL	RO, END_OF_SIDR		1311
		50	D4	00167		CLRL	-(SP)		
		7E	D4	00169		BSBW	RMS\$IDR_FIRST		
		0000G	30	0016C		ADDL2	#4, SP		
		5E	C0	0016C		MOVL	RO, REC_ADDR		
		56	D0	0016F		CLRW	118(IRAB)		
		76	A9	B4	00172	BRW	3\$		1312
		FED1	31	00175		BLBS	STATUS, 22\$		1091
32		10	AE	E8	00178	18\$:	TSTL	32(IRAB)	1329
		20	A9	D5	0017C		BEQL	19\$	1336
		OF	13	0017F		MOVL	32(IRAB), BDB		
		54	20	A9	D0	00181	CLRL	32(IRAB)	1338
		20	A9	D4	00185		CLRL	-(SP)	
		7E	D4	00188		BSBW	RMS\$RLSBKT		
		0000G	30	0018A		ADDL2	#4, SP		
		5E	C0	0018D		CMPW	STATUS, #33458		
		8F	10	AE	B1	00190	BNEQ	20\$	1340
			OB	12	00196		MOVL	STATUS, 12(RAB)	
0C	A8	10	AE	D0	00198		MOVZWL	#33844, STATUS	1343
10	AE	8434	8F	3C	0019D		BICB2	#128, 66(IRAB)	1344
42	A9	80	8F	8A	001A3	20\$:	MOVL	STATUS, RO	1347
		50	10	AE	D0	001A8	BRB	24\$	1357
			40	11	001AC	21\$:	MOVL	60(IRAB), BDB	
		54	3C	A9	D0	001AE	CLRL	60(IRAB)	1365
		3C	A9	D4	001B2	22\$:	MOVL	28(BDB), 112(IRAB)	1366
70	A9	1C	A4	D0	001B5		MOVL	REC_ADDR, SAVE_REC_ADDR	1368
		53	56	D0	001BA		MOVL	BEG_OF_SIDR, REC_ADDR	1370
		56	04	AE	D0	001BD	PUSHL	#1	1371
			01	DD	001C1		BSBW	RMS\$IDR_FIRST	1372
		5E	0000G	30	001C3		ADDL2	#4, SP	
		A9	04	C0	001C6		MOVL	R1, 124(IRAB)	
0082	C9		51	D0	001C9		MOVW	R2, 130(IRAB)	
		56	52	B0	001CD		MOVL	SAVE_REC_ADDR, REC_ADDR	1373
			53	D0	001D2		TSTB	66(IRAB)	1376
		42	A9	95	001D5		BGEQ	23\$	
			04	18	001D8		BISB2	#8, 10(BDB)	1378
0A	A4	08	88	001DA		CLRL	-(SP)		1380
		7E	D4	001DE	23\$:	BSBW	RMS\$RLSBKT		
		0000G	30	001E0		ADDL2	#4, SP		
		5E	04	C0	001E3		BICB2	#128, 66(IRAB)	1381
42	A9	80	8F	8A	001E6		MOVL	#1, RO	1383
		50	01	D0	001EB		ADDL2	#28, SP	1386
		5E	1C	C0	001EE	24\$:	POPR	#^M<R2,R3,R4,R5>	
		3C	BA	001F1		RSB			
			05	001F3					

: Routine Size: 500 bytes, Routine Base: RMSRMS3 + 0160

1326	1387	1
1327	1388	1
1328	1389	1
1329	1390	0

END
ELUDOM

RM3SSIDR
V04-000

RMSSEARCH_SIDR

H 11
16-Sep-1984 02:07:52
14-Sep-1984 13:01:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[RMS.SRC]RM3SSIDR.B32;1

Page 31
(4)

RM
VC

PSECT SUMMARY

Name	Bytes	Attributes
RMSRMS3	852	NOVEC,NOWRT, RD , EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	----- Symbols -----			Pages Mapped	Processing Time
	Total	Loaded	Percent		
\$_\$255\$DUA28:[RMS.OBJ]RMS.L32;1	3109	70	2	154	00:00.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:\$RM3SSIDR/OBJ=OBJ\$:\$RM3SSIDR MSRC\$:\$RM3SSIDR/UPDATE-(ENH\$:\$RM3SSIDR)

: Size: 852 code + 0 data bytes
: Run Time: 00:23.7
: Elapsed Time: 00:48.1
: Lines/CPU Min: 3513
: Lexemes/CPU-Min: 14567
: Memory Used: 195 pages
: Compilation Complete

032B AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RM3551DR
LIS

RM3UPSIDX
LIS

RM35RCHKY
LIS

RM3UPDDEL
LIS

RM3UPDATE
LIS